An Algorithm for Optimal Partitioning of Data on an Interval

Brad Jackson, Jeffrey D. Scargle, David Barnes, Sundararajan Arabhi, Alina Alt, Peter Gioumousis,

Elyus Gwin, Paungkaew Sangtrakulcharoen, Linda Tan, and Tun Tao Tsai

Manuscript received XXX, 2003. This work was supported by the NASA Applied Information Systems Research Program, the Woodward Fund of San Jose State University, and the NASA Faculty Fellowship Program at Ames Research Center and Dryden Flight Research Center.

B. Jackson is with the Department of Mathematics, San Jose State University, J. Scargle is with the Space Science Division, NASA Ames Research Center. The other authors were participants in the Center for Applied Mathematics and Computer Science program at San Jose State University.

Corresponding author:
Jeffrey D. Scargle, MS 245-3
Space Science Division
NASA Ames Research Center
Moffett Field, CA, 94035-1000
Telephone: 650 604 6330
Fax: 650 604 -6779
E-mail: Jeffrey.D.Scargle@nasa.gov

Subject headings:

signal detection, density estimation, optimization, Bayesian modeling, histograms, cluster analysis

EDICS: 1.STAT

1. Abstract

Many signal processing problems can be solved by maximizing the fitness of a segmented model over all possible partitions of the data interval. This letter describes a simple but powerful algorithm that searches the exponentially large space of partitions of N data points in time $O(N^2)$. The algorithm is guaranteed to find the exact global optimum, automatically determines the model order (the number of segments), has a convenient real-time mode, can be extended to higher dimensional data spaces, and solves a surprising variety of problems in signal detection and characterization, density estimation, cluster analysis and classification.

2. Introduction: The Problem

A variety of signal processing and related problems can be viewed as the search for an optimal partition of data given on a time interval I. For example, one may estimate a segmented model by maximizing some measure of model fitness[1] defined on partitions of I. Since the space of all partitions defined by a continuum of subintervals is highly infinite, it is advantageous to discretize the interval. Often the data themselves,

$$X_n, n = 1, 2, \ldots, N, \tag{1}$$

where N is the number of data points, naturally subdivide I into into connected subintervals – which we call *data cells*. We avoid a specific definition of the data cells, because many different types are possible. Common examples are counts in bins, measurements at a set of sample times (evenly spaced or not), and event or point data. The underlying idea is that restricting consideration to the finite space of partitions whose elements are sets of data cells will result in no significant loss of information or of resolution in the independent variable.

A *partition* $\mathbf{P}$ of an interval I is a set of M *blocks*,

$$\mathbf{P}(I) = \{B_m, m \in \mathcal{M}\}\ , \mathcal{M} \equiv \{1, 2, \ldots M\}\ , \tag{2}$$

where the blocks are sets of data cells defined by index sets $\mathcal{N}_m$:

$$B_m = \{X_n, n \in \mathcal{N}_m\} \tag{3}$$

satisfying the usual conditions, $\bigcup_m B_m = I$ and $B_m \bigcap B_{m'} = \emptyset$ if $m \neq m'$. Partitions will be denoted in boldface, and refer to the interval I unless otherwise stated. Define P^* as the (finite) set of all possible partitions of I

[1]This concept goes by many names, including goodness of fit, loss, penalty, objective function, risk *etc.*, but here we use the term *cost*.

into connected blocks. Take as given an additive *cost function* that assigns a value to any partition $\mathbf{P} \in P^*$ in the form

$$V(\mathbf{P}) = \sum_{m=1}^{M} g(B_m) \ , \tag{4}$$

where $g(B_m)$ is the cost of block B_m. Computationally, the data cells must be represented by a data structure that contains *sufficient statistics* for the model – *i.e.* all information necessary to determine g for any block [see *e.g.* Eq. (11)].

We exhibit an efficient $O(N^2)$ dynamic programming algorithm that finds an *optimal partition* $\mathbf{P}^{\max} \in P^*$: $V(\mathbf{P}^{\max}) \geq V(\mathbf{P})$ for all partitions $\mathbf{P} \in P^*$.

Scargle [5] proposed two greedy iterative algorithms for finding near-optimal partitions: one top-down (optimally divide I into two parts; recursively do the same to each such part) the other bottom-up (merge adjacent data cells). In both cases Bayesian model comparison provides effective fitness functions and halting criteria, implementing an $O(N^2)$ procedure for data spaces of 1, 2 and higher dimensions – hence the term *Bayesian Blocks* [5]. But in practice these greedy algorithms often find significantly suboptimal partitions, motivating the development reported here.

3. Dynamic Programming: Finding Optimal Partitions

We describe an $O(N^2)$ algorithm that is guaranteed to solve the above problem by finding an exact global optimum, for any cost function V that is additive in the sense of Eq. (4). There is a large (2^{N-1}) but finite number of partitions in P^*. Dynamic programming [2] is an intelligent method of searching this space of all possible solutions of our optimization problem. It can be applied whenever the *principle of optimality* – in this context, any subpartition of an optimal partition is optimal – holds.

Theorem 1 (Principle of Optimality) *Let* $\mathbf{P}^{\max}$ *be an optimal partition of* I *and* $\mathbf{P}_1 = \{B_m, m \in a\}$ *be any subset of the blocks of* $\mathbf{P}^{\max}$. *Then* $\mathbf{P}_1$ *is an optimal partition of the part of* I *it covers, namely* $I_1 = \bigcup_{m \in a} B_m$.

Intuitively, this result follows from the fact that a better subpartition could be used to construct a partition of the full interval better than the optimal one – a clear contradiction. The proof relies on the fact that the block-additivity of the cost function implies that it is also additive on subpartitions. To see this, divide partition $\mathbf{P}$ into any two disjoint parts, $\mathbf{P}_1 = \{B_m, m \in a\}$ and $\mathbf{P}_2 = \{B_m, m \in b\}$, with $\mathbf{P}_1 \bigcup \mathbf{P}_2 = \mathbf{P}$ and $a \bigcup b = \mathcal{M}$. Then the additivity of V yields

$$\begin{aligned} V(\mathbf{P}) &= \sum_{m=1}^{M} g(B_m) \\ &= \sum_{m \in a} g(B_m) + \sum_{m \in b} g(B_m) \end{aligned}$$

$$= \quad V(\mathbf{P}_1) + V(\mathbf{P}_2). \tag{5}$$

Proof 1: As above, denote by $\mathbf{P}_2$ the subpartition of $\mathbf{P}^{\max}$, consisting of the blocks $\{B_m, m \in \mathcal{M} - a\}$ in $\mathbf{P}^{\max}$ that are not in $\mathbf{P}_1$. Let $\mathbf{P}_3$ be any other partition of I_1. Since $\mathbf{P}^{\max}$ is an optimal partition of I and $\mathbf{P}_3 \cup \mathbf{P}_2$ is also a partition of I it follows that $V(\mathbf{P}^{\max}) = V(\mathbf{P}_1) + V(\mathbf{P}_2) \geq V(\mathbf{P}_3 \cup \mathbf{P}_2) = V(\mathbf{P}_3) + V(\mathbf{P}_2)$ so $V(\mathbf{P}_1) \geq V(\mathbf{P}_3)$. Thus $\mathbf{P}_1$ is an optimal partition of I_1.

Dynamic programming is a recursive procedure that can be used to efficiently find the solution to many kinds of combinatorial optimization problems. Our algorithm derives the optimal partition of the first $n+1$ data points using that of the first n. At each iteration we must consider all possible starting locations j, $1 \leq j \leq n$ of the last block of the optimal partition. For each putative j the cost function is – by the principle of optimality – the cost of the optimal subpartition prior to j plus the cost of the last block itself. The former was stored at previous iterations, and the latter is a simple evaluation of V. The desired new optimal partition corresponds to the maximum over all j.

More precisely, define $opt(n)$ to be the value of the cost function of the optimal partition $\mathbf{P}_n^{\max}$ of the first n cells of I, for $1 \leq n \leq N$. The following dynamic programming algorithm finds the optimal partition $\mathbf{P}_N^{\max}$:

1. Define $opt(0) = 0$
2. Given that $opt(j)$ has been determined for $j = 0, 1, \ldots, n$:
   - Define $end(j, n+1) = g(B_{j,n+1})$; $B_{j,n+1}$ is the union of cells $j, j+1, \ldots, n+1$
   - Then compute
   $$opt(n+1) = \underset{j}{\text{Max}}\{opt(j-1) + end(j, n+1)\}, \tag{6}$$
   for $j = 1, 2, \ldots, n+1$.
   - The value of j where this maximum occurs is stored as $lastchange(n+1)$.
3. Repeat 2 until $n+1 = N$, when $opt(N)$, the optimal partition cost for all N cells, has been obtained.
4. Backtrack using the *lastchange* vector to identify the start points of individual blocks of the optimal partition $\mathbf{P}^{\max}$ in the following way. Let $n_1 = lastchange(N), n_2 = lastchange(n_1 - 1)$, *etc.* Then the last block in $\mathbf{P}^{\max}$ contains cells $n_1, n_1 + 1, \ldots, N$, the next-to-last block in $\mathbf{P}^{\max}$ contains cells $n_2, n_2 + 1, \ldots, n_1 - 1$, and so on.

Theorem 2 *This deterministic $O(N^2)$ dynamic programming algorithm finds the partition of I that maximizes the (additive) cost function.*

Proof 2: The proof is by mathematical induction. Clearly $opt(1) = \text{Max}\{0 + end(1,1)\} = g(B_{1,1})$ is the cost of the only possible (and therefore optimal) partition of the set comprising the first cell. At iteration $n+1$, assume not only that we have found the optimum partition of $\mathbf{P}_n^{\max}$, but also that for $i = 1, 2, \ldots, n$, we have stored the corresponding cost for this and all previous iterations in the array $opt(i)$, and the index of the cell beginning this partition's last block in array $lastchange(i)$. Let $F(j) = opt(j-1) + end(j, n+1)$; then the principle of optimality shows that when j indexes the first cell of the last block of the desired partition $\mathbf{P}_{n+1}^{\max}, F(j)$ is the corresponding maximum cost. Further, for any $j, F(j)$ is the cost of a legitimate parition of $\mathbf{P}_{n+1}^{\max}$, namely that consisting of the optimal partition of the cells prior to j followed by the single block $B_{j,n+1}$. These two facts combine to prove that the maximum of $F(j)$ specified in Eq. (6) gives the desired optimum partition at iteration $n+1$. Identification of the corresponding optimal blocks – starting with the last one and working backwards, as in part (4) of the algorithm – can be validated with straightforward recursive application of the principle of optimality. Finally, since $end(j, n+1) = g(B_{j,n+1})$ the algorithm requires $1 + 2 + \ldots + N = O(N^2)$ evaluations of the function g. It also requires $O(N^2)$ additions and $O(N^2)$ comparisons in determining the maximums.

4. Applications

These results apply to any segmented modeling of 1-dimensional data. As a key density estimation example, piecewise constant models yield histograms in which the bins are not constrained to be equal. The number of bins and their sizes and locations are determined by the data. Further, almost all of the results described here can be easily extended – almost without change – to data of higher dimensionality, as will be described in future papers [4]. Cluster analysis, not usually considered for one-dimensional data, can be effected as a post-processing of segmented models – piecing the blocks together into clusters – and similarly with unsupervised classification and other data mining procedures.

But the following problem, relevant to any signal detection and charactarization problem with event data, has been our primary application.

In many astronomical observations individual photons emitted by a source are detected. The data stream consists of a list of the corresponding detection times, one for each photon. The astrophysical goal is to detect and characterize the variations (if any) of the intensity of the source. For example, if there is a significant outburst of radiation, when does it start and what is its intensity? Both of these questions can be answered by finding the step function that best fits the data.

Here I is the time interval during which the source was observed. There is a point on I for each *event*, marking the time that the photon was detected. Many methods of analyzing such data require *binning* of

the data. Usually the bins are taken be large enough so that they each contain enough photon counts to provide a good statistical sample. This practice of binning event data throws away a considerable amount of information and introduces a dependency of the results on the sizes and locations of the bins. Our method does not require bins.

We start with the subdivision of the observation interval into data cells, as described above. For event data, the midpoints between successive data points provide a suitable definition of the cell edges. This can be thought of as a 1D Voronoi tessellation[2] of the data, a view that extends naturally to higher dimensions. Even though the reciprocal of the cell size is a useful estimate of the local density, one is not usually satisfied with this representation. It comprises N blocks and is much too refined. It also reflects the noise inherent in the data. Consider partitions $\mathbf{P}$ of I into blocks that are connected unions of these cells. For any block $B_m \in \mathbf{P}$, we denote its length by a_m and the number of events in it by N_m. This representation is conveniently plotted as a step function with M steps, the width of the mth step is a_m and the height of the mth step is the corresponding event density N_m/a_m.

It is necessary to specify the fitness measure to be maximized in finding the optimal model. Many such cost functions are possible, but we have found the Bayesian posterior for a segmented Poisson model very useful in this and related contexts. That is, each block is modeled as a Poisson process with constant intensity, yielding for the posterior block probability [5]:

$$Pr(B_m|D_m) = \frac{\Gamma(N_m+1)\Gamma(a_m-N_m+1)}{\Gamma(a_m+2)} \ , \tag{7}$$

where Γ is the standard gamma function. D_m represents the data in the block, expressed in terms of two *sufficient statistics* a_m and N_m. The Poisson intensity parameter has been marginalized, using a flat prior. In many applications a_m is an integer, so the corresponding factorial functions can be used.

The posterior probability of a given partition is the product of the posteriors over all the blocks in that partition, since we assume the points in each interval are independent of each other. For a Poisson process this holds if there are no correlations in the detections of the events. In practice, detectors have a small dead time - that is, for a short time after detecting a photon no further events can be detected. But here we neglect this effect. Thus the best (most likely) partition is one which maximizes

$$Pr(\mathbf{P}|D) = \prod_{m=1}^{M} Pr(B_m|D) \ , \tag{8}$$

where the product is taken over all the blocks in the partition. Equivalently we can maximize

$$V_{events} = \sum_{m=1}^{M} \left[\log\Gamma(N_m+1) + \log\Gamma(a_m-N_m+1) - \log\Gamma(a_m+2)\right] . \tag{9}$$

[2]The *Voronoi tessellation* divides a data space S into cells such that each point in S is assigned to the cell containing the data point that it is closest to.

For binned data 5

$$V_{bins} = \sum_{m=1}^{M} [\log\Gamma(N_m + 1) - (N_m + 1) log(a_m + 1)] \ . \tag{10}$$

For the model consisting of a signal with additive Gaussian noise, the data cell contains the measured value, the time of measurement, and the standard deviation of the noise:

$$\mathbf{X}_n = \{x_n, t_n, \sigma_n\} \ \ n = 1, 2, \ldots, N \ , \tag{11}$$

and for a flat (unnormalized) prior it is easy to derive the posterior:

$$V_{normal} = \sum_k [-\frac{1}{2}\log(a_k) + (\frac{b_k^2}{4a_k}) - c_k] \tag{12}$$

where

$$a_k = \frac{1}{2}\sum_n \frac{1}{\sigma_n^2} \tag{13}$$

$$b_k = -\sum_n \frac{x_n}{\sigma_n^2} \tag{14}$$

and

$$c_k = \frac{1}{2}\sum_n \frac{x_n^2}{\sigma_n^2} \tag{15}$$

To treat the the number of blocks as a nuisance parameter, we assign a geometric prior [3], which adds a term proportional to the number of blocks into the final expression for V to be maximized.

Analysis of arrival times of photons detected from a gamma-ray burst observed by the NASA BATSE instrument on the Compton Gamma-ray Observatory is shown in Figure 1. The only adjustable parameter – from the the geometric prior described above – was chosen based on simulation studies on synthetic data with properties similar to the BATSE data.

As we have seen dynamic programming gives a good (polynomial) algorithm for finding an optimal partition of data on an interval for any cost function V satisfying the additive property [see Eq.(4)]. Ironically it has the same $O(N^2)$ complexity as the greedy algorithm. Note that Vidal [6] gives an $O(kN^2)$ dynamic programming algorithm for finding the optimal partition of an interval into k blocks, for a given k.

5. Conclusion

In comparing the use of our algorithm to detect and characterize clusters (collections of blocks) with some of the standard clustering techniques [1], we note that our method inherently compares partitions that have different numbers of blocks, so the number of blocks is automatically determined by the data. This is to be contrasted with most standard clustering techniques, in which k, the fixed number of clusters must be specified ahead of time. One often seeks to minimize the maximum diameter (defined as the maximum

distance between any pair of points in the cluster) of the clusters, or to maximize the minimum separation between the clusters. In dimension 1, there are well-known $O(kN^2)$ dynamic programming algorithms for finding the best partitions into k clusters. For dimension 2 and higher it is known that these standard problems are NP-complete. We don't yet know if our problem is NP-complete in dimension 2 and higher.

In addition, considered as a density estimation or signal detection technique, our approach does not introduce any explicit smoothing of the data. Structure on any time scale, no matter how short, will be detected if it is supported by the data. While the parameter in the geometric prior discussed above contols to some extent the number of blocks – and thus affects the roughness of the optimized model – it is not explicitly a smoothing parameter. Another feature is that the incremental way the algorithm operates on the data makes a real-time mode trivial to implement. This mode has found to be very useful in the rapid detection of changepoints in a data stream. And since $opt(i+1)$ is calculated from $opt(j), j = 1,2,\ldots,i$, some of the necessary calculations can be performed as the data are still being collected. Also it is easy to modify the dynamic programming to yield the optimal partition with blocks of a minimum size (each block contains at least d data points, for a given positive integer d). These and other features are described in more detail at an algorithm repository at: `http://trotsky.arc.nasa.gov/~pgazis/CodeArchiveServer/CodeArchiveServer.html`

REFERENCES

[1] C. J. Alpert and A. B. Kahng , Splitting Orderings into Multi-way Partitionings to Minimize the Maximum Diameter, Journal of Classification, (14), 1997, pp. 51-74.

[2] R. Bellman, *Dynamic Programming*, Princeton University Press, Princeton, 1957.

[3] M. A. Coram *Nonparametric Bayesian Classification*, Ph.D. thesis, Department of Statistics, Stanford University, 2002.

[4] B. Jackson, J. Scargle, et. al., *Optimal Partitions of Data in Higher Dimensions*, in preparation.

[5] J. Scargle, *Studies in Astronomical Time Series Analysis. V. Bayesian Blocks, A New Method to Analyze Structure in Photon Counting Data*, The Astrophysical Journal, (504), 1998, pp. 405-418.

[6] R. Vidal, *Optimal Partition of an Interval*, Applied Simulated Annealing, Springer-Verlag, New York, 1993, pp. 291-314.

This manuscript was prepared with the AAS LaTeX macros v4.0.

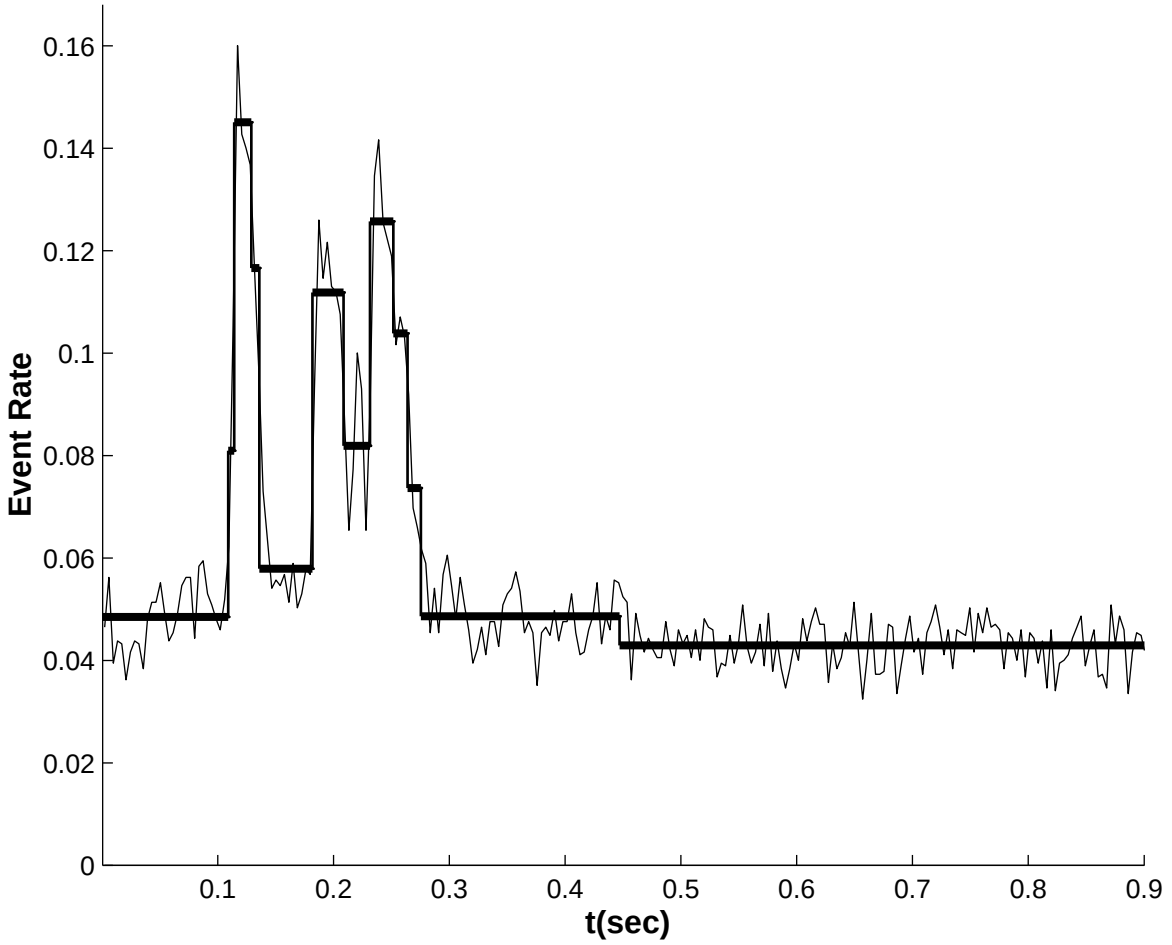

Fig. 1.— Bayesian Block analysis of event data, namely arrival times of 25,213 gamma-ray photons detected from gamma-ray burst Tr0551 by the BATSE instrument on the NASA Compton Gamma-ray Observatory. Event rate (counts per unit time) is plotted as a function of time, in seconds. The thin line shows the counts in evenly spaced bins of width .002 sec, chosen simply to display the raw data. The solid lines show the optimal partition into blocks as determined with our algorithm.